UNIVERSITÀ DEGLI STUDI DI ROMA
"LA SAPIENZA"

DIPARTIMENTO DI
INFORMATICA E SISTEMISTICA

A. PAOLUZZI

# A ROBUST, TILE-BASED ALGORITHM FOR POINT/POLYGON CLASSIFICATION

RAP. 03.86 GIUGNO 1986

A. PAOLUZZI

# A ROBUST, TILE-BASED ALGORITHM FOR POINT/POLYGON CLASSIFICATION

Alberto Paoluzzi: Dipartimento di Informatica e Sistemistica, Universita' "La Sapienza", Via Buonarroti 12, I-00185, Roma, Italy, (39)-6-7312367/7312328.

ABSTRACT

In this paper a very efficient and robust linear algorithm is given for deciding if a point is internal, external or on the boundary of a general polygon, also no simple and no connected. The proposed algorithm is based on the technique of counting the number of intersections between the polygon boundary and a ray starting from the test point. According to the proposed method, no arithmetic operations are usually needed to perform the test, because mainly comparison operations are used. The algorithm is also caracterized by a probabilistic analisys of the set of possible inputs, which are classified in disjoint subsets of different expectation. As a consequence of its robustness and efficiency, it appears to be particularly useful for actual implementations.

**Categories and Subject Descriptors:**

F.2.2. [Analisys of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems - geometrical problems and computations;
I.3.5. [Computer Graphics]: Computational geometry and Object Modeling;

**General Terms:** Algorithms, Design, Theory.

**Additional Keywords and Phrases:** tile code, polygon, containment test, point/set membership, point enclosure problem.

## 1. INTRODUCTION

In computational geometry and object modeling areas it is often necessary to search for which part of a set X is internal, external or on the boundary of a given set A [1]. A similar problem is posed by the question: "is a point p internal, external or on the boundary of a given set A?" In Computer Graphics frameworks this question is usually called "containment test" [2]; other autors refer to this problem as a "point enclosure problem" [3]. If the set A is a planar linear polygon, we shall call this test, following Tilove [1], a "point/polygon classification".

Point/polygon classification is very frequently needed as a basic operation in Computer Graphics, Solid Modeling and Pattern Recognition algorithms: consider, for instance, the implementation of pick input primitive in interactive graphic packages, hidden line/surface elimination problems, boolean operations over polygons and polyedra, etc. It is well-known to people working in the Computer Graphics field, that the main problem in actual implementations of point/polygon classification algorithms concerns their geometrical robustness, i.e. their behaviour for any possible (and strange) input.

A point/polygon classification function Class(p,A) is defined if and only if p belongs to the same plane of A, and assumes its values onto the set {p.in.A, p.on.A, p.out.A}. By means of a linear transformation, eventually performed by using homogeneous coordinates, the plane of A and p can always be transformed onto the plane z = 0. For this reason we will refer the algorithm to the bidimensional case.

Two different techniques are generally used for identifying the relative position of a general linear polygon and of a point belonging to the same plane. The first technique consists of the algebraic summation of the angles under which the polygon edges are seen from the given point. If this sum is zero the point is external to the polygon; otherwise the point is internal or on the boundary. In this case a further step must be performed to decide between .in. and .on. values for the classification function. Conversely, the second technique is based on a well-known topological property: a point is either internal or external with respect to a given closed curve, according to whether the number of intersections between the curve and any ray starting at the point is respectively odd or even.

Both these techniques have a computational performance of O(n) query time, when n is the number of polygon vertices (and edges) [4]. Better algorithms are known for the convex case [5]. Nevertheless, even if linear, algorithms for the general case are enough expensive, because both types require arithmetic operations (multiplications and divisions) for every edge of the polygon. From this point of view, the intersection counting technique is a little better than the one based on angles.

In this paper we present a new version of the counting method, giving an algorithm based on the detection more than on the computation of the intersection points. This algorithm is based on a complete classification of every possible relative position of the query point and of an edge of the polygon boundary, and requires, in the most of cases, only few logical comparisons for every edge. The explicit execution of a line intersection operation (corresponding to one multiplication and to one division) is required only a number of times which is very smaller than the number of effective intersection points between the polygon boundary and the ray for the query point, as we will see at section 4.

The algorithm makes also an extensive use of the "tile code" of a point, defined by Cohen and Sutherland in the sixties and recently used [6] for solving the Polygon Clipping Problem.

The main feature of the proposed algorithm is its geometrical robustness: it works well also with no simple and no connected polygons, with polygons having repeated vertices and edges, and doesn't present hard numerical or geometrical problems when the query point coincides with vertices or belongs to edges.

The paper is organized as follows: in section 2 we briefly recall basic concepts on tile codes and on the computation of segment intersection; in section 3 the concept of "edge code" is defined and the graphical framework of the algorithm is outlined; in section 4 are presented the main geometrical ideas of the paper. In section 5 the algorithm is formally given and finally a probabilistic analisys of its performance is attempted. The appendices contain a simple Pascal implementation of the algorithm and the discussion of two numerical cases.

## 2. PRELIMINARIES

### 2.1 tile code

A rectangular domain parallel to the axes of the coordinate system can be defined as the intersection of the four subspaces:

$y \leq ymax$; $x \leq xmax$;

$y \geq ymin$; $x \geq xmin$.

The border lines of these subspaces divide the plane in nine regions, eigth of which are unbounded. Such regions can be viewed as a "tile array", being each element of the array labeled by a tile code constituted by the subset of the identificators of subspaces "violated" by the corresponding region. In the following we shall use the coding schema shown in figure 1.

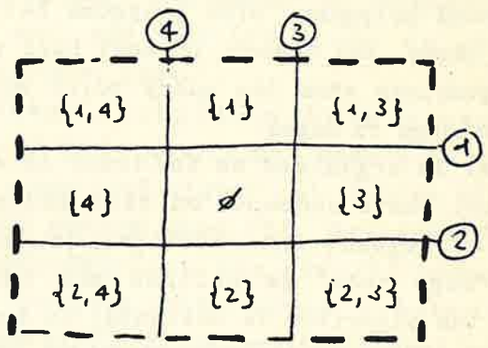



**figure 1.** Tile array and coding schema used in the paper.

### 2.2 segment intersection

Two straigth line segments (in the following denoted simply as "segments") can have either one, zero or an infinite number of common points. The search for common points is usually performed by verifying if both segments contain the common points of their infinite lines.

In our algorithm we are interested in calculating, if it exists, the intersection point between a given segment, having extremes denoted as $p_1$ and $p_2$, and the horizontal semi-infinite line starting at test point p. In resolving the system formed by the equation $y = y_p$ and by the parametric equation of the line for $p_1$ and $p_2$, the intersection point between infinite lines exists and is unique if and only if $y_1 \neq y_2$. In this case we have for the parameter: $t = (y_1 - y_p)/(y_1 - y_2)$.

The calculated point $(1 - t)\, p_1 + t\, p_2$ belongs to the segment $p_1p_2$ if $0 \leq t \leq 1$. However, this point belongs also to the rigth semi-infinite horizontal line starting at p only if: $x_p \leq (1 - t)\, x_1 + t\, x_2$.

## 3. GEOMETRICAL FRAMEWORK

In the following we shall consider separately every edge of the given polygon, and denote it as $\langle p_i, p_{i+1} \rangle$. Our objective is to determine, with the minimum computational effort, if this segment intersects the semi-infinite line for the query point.

The algorithm proposed is centered on the computation of tile codes of vertices, with respect to a special tile array T(p) having its central element coincident with the query point p. In other words, we pose for the border lines of tile array:

$x_{min} = x_{max} = x$; $y_{min} = y_{may} = y$,

being $p = [x \;\; y]$. In this way we get a special tile array T having the elements $t_{12}$, $t_{32}$, $t_{21}$, $t_{23}$ representing

linear domains coincident with four orthogonal semi-infinite lines starting at p, and having also the element $t_{22}$ coincident with the test point p . See figure 2 for a pictorial representation of the corresponding partition of the plane.

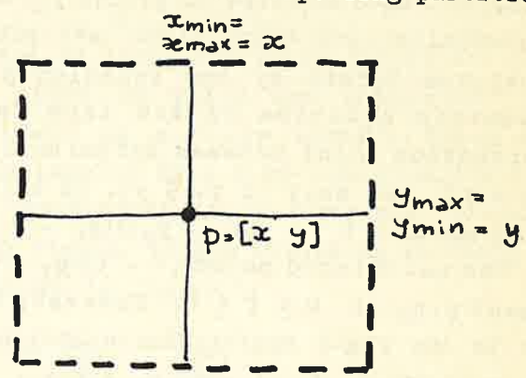



**Figure 2.** Special tile array centered on test point p.

For clarity sake, in the following we will continue to draw our special tile array as having a row and a column of non null dimensionality; we must remember that the two bounds of them actually coincide (see figure 3).

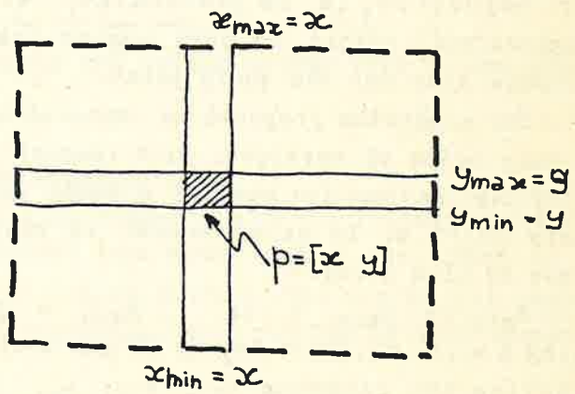



**Figure 3.**

We will call **edge-code** of an edge e = $\langle p_1, p_2 \rangle$, and denote it as $c_e$, the subset of the four boundary lines of the current tile array so defined:

$$c_e = (c_1 \quad c_2) - (c_1 \quad c_2),$$

being $c_1$ and $c_2$ the tile codes of edge extreme points. It is very easy to see that an edge-code represents the subset of tile border lines intersected by the considered edge [6].

## 4. CLASSIFICATION OF POSSIBLE INPUTS

In this section we classify polygon edges in disjoint classes, in function of their position relative to the query point p. For every class of edges we also set up a minimum set of computations in order to detect the number of intersections between edges of the class and the horizontal ray for p.

Let p be the query point; it determines a partition T(p) of the plane in nine sectors (tiles), four of which of dimension 2 (isomorphs to $\mathbf{R}_2$), four of dimension 1 (isomorphs to $\mathbf{R}_1$), and one of dimension 0 (the point p).

Given a test point p and a partition T(p) of the plane, we wont to calculate the number of possible different position of an edge with respect to the partition. Because every extreme of the edge can fall into anyone of the nine sectors of the plane, we should have 9x9 = 81 different positions of an edge into the plane partition T(p). But, in order to calculate the classification function Class(p, A) by using the inter-

section counting method, it is not necessary to consider the orientation of edges, i.e. to distinguish the couple $\langle p_1, p_2 \rangle$ from the couple $\langle p_2, p_1 \rangle$. For this reason it is possible to identificate all symmetrical cases, and consider only $(81 - 9)/2 + 9 = 45$ different relative positions for a test point and a polygon edge.

A first level classification of the 45 possible positions of an edge can be done by computing its edge-code, which can take anyone of the 16 values in $2^{\{1,2,3,4\}}$.

In the following we analyse, for any value of edge-code, the possible relative positions of edges and of the ray for the test point. For any case we give also subsidiary logical conditions which allow to directly detect the number of intersections n(e,p) between the edge e and the horizontal ray for p. We will assume:

a) $n(e,p) = 0$ if the edge and the ray don't intersect or if they have infinite common points;

b) $n(e,p) = 1$ if the intersection point is unique and is different from the extreme points $p_1$ and $p_2$;

c) $n(e,p) = \pm 0.5$ if the intersection point coincide with $p_1$ or $p_2$; the sign is dependent from the position of preceeding edges;

The reasons for positions a) b) and c) are pictorially explained in figures 4 and 5: if the boundary of the polygon effectively crosses the ray for p we have to add 1 to the intersection counter, otherwise we have to add 0.

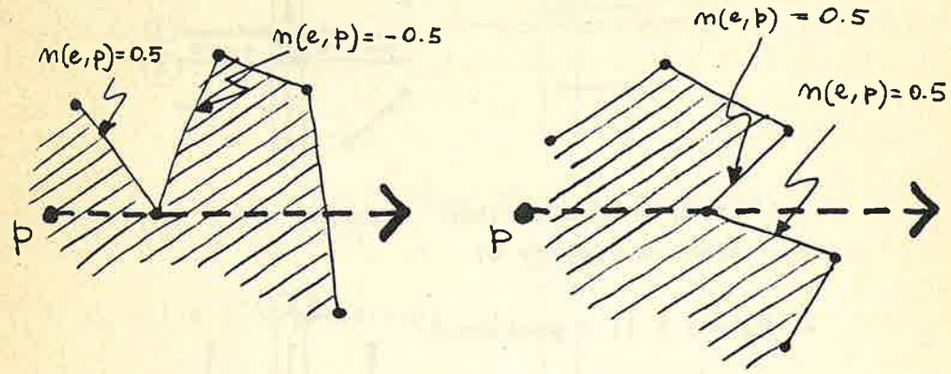



**Figure 4.** Edges having a vertex on the ray

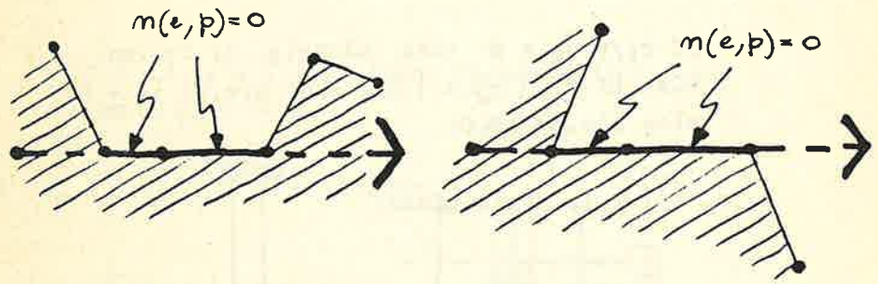
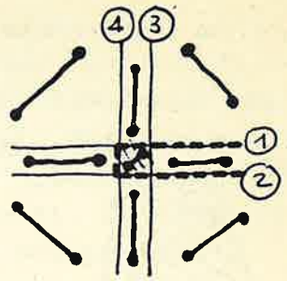



**Figure 5.** Edges lying on the ray.

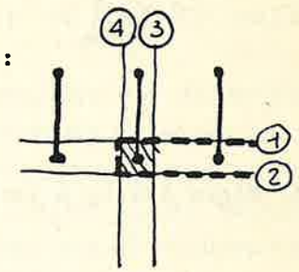
In the following the possible 45 relative positions of an edge and of a test point are grouped into 16 classes, depending on the value of the edge-code.

1. $c_e = \emptyset$; 9 positions:



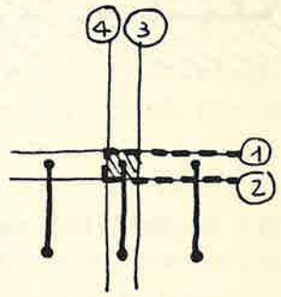

if $c_1 \cup c_2 = \emptyset$ then Class(p, A) := .on.
else n(e,p) := 0;

2. $c_e = \{ 1 \}$; 3 positions:



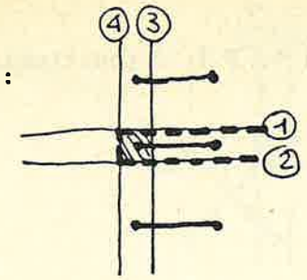

if $c_1 \cap c_2 = \emptyset$ then Class(p, A) := .on.
else if $c_1 \cap c_2 = \{ 3 \}$ then n(e,p) := $\pm$ 0.5
else n(e,p) := 0;

3. $c_e = \{ 2 \}$; 3 positions:



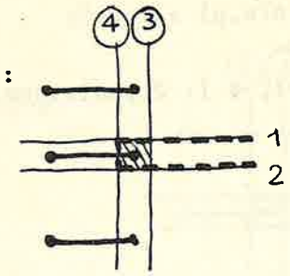

if $c_1 \cap c_2 = \emptyset$ then Class(p, A) := .on.
else if $c_1 \cap c_2 = \{ 3 \}$ then n(e,p) := $\pm$ 0.5
else n(e,p) := 0;

4. $c_e = \{ 3 \}$; 3 positions:



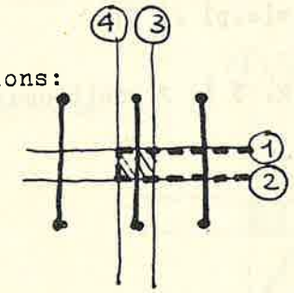

if $c_1 \cup c_2 = \{ 3 \}$ then Class(p, A) := .on.
else n(e,p) := 0;

5. $c_e = \{ 4 \}$; 3 positions:



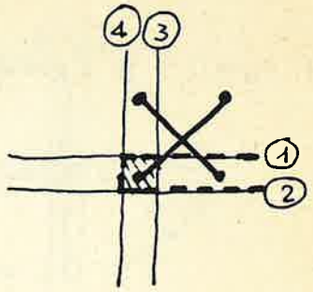

if $c_1 \cup c_2 = \{ 4 \}$ then Class(p, A) := .on.
else n(e,p) := 0;

6. $c_e = \{ 1, 2 \}$; 3 positions:



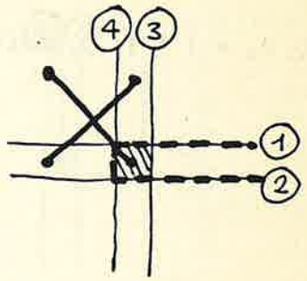

if $c_1 \cap c_2 = \emptyset$ then Class(p, A) := .on.
else if $c_1 \cap c_2 = \{ 3 \}$ then n(e,p) := 1
else n(e,p) := 0;

7. $c_e = \{ 1, 3 \}$; 2 positions:



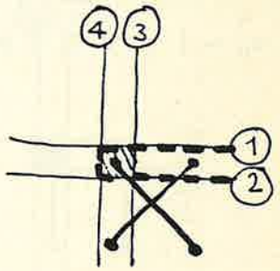

if $(c_1=\emptyset)$ or $(c_2=\emptyset)$ then Class(p, A) := .on.
else n(e,p) := $\pm$0.5;

8. $c_e = \{ 1, 4 \}$; 2 positions:



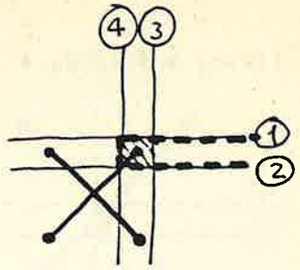

if $(c_1=\emptyset)$ or $(c_2=\emptyset)$ then Class(p, A) := .on.
else n(e,p) := 0;

9. $c_e = \{ 2, 3 \}$; 2 positions:



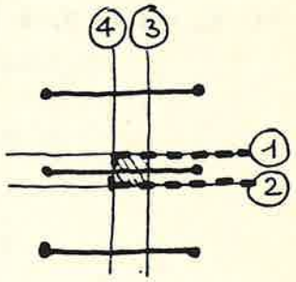

if $(c_1=\emptyset)$ or $(c_2=\emptyset)$ then Class(p, A) := .on.
else n(e,p) := $\pm$0.5;

10. $c_e = \{ 2, 4 \}$; 2 positions:



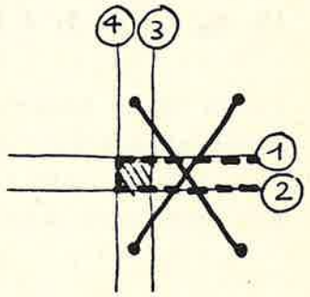

if $(c_1=\emptyset)$ or $(c_2=\emptyset)$ then Class(p, A) := .on.
else n(e,p) := 0;

11. $c_e = \{ 3, 4 \}$; 3 positions:



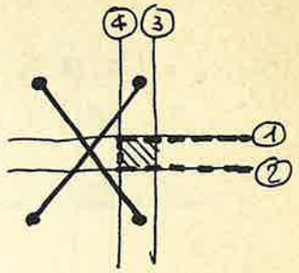

if $c_1 \cup c_2 = \{ 3, 4 \}$ then Class(p, A) := .on.
else n(e,p) := 0;

12. $c_e = \{ 1, 2, 3 \}$; 2 positions:



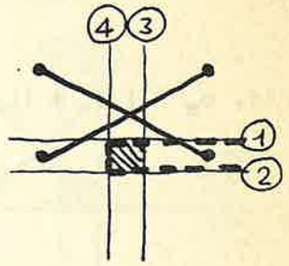

no subsidiary test: n(e,p) := 1;

13. $c_e$ = { 1, 2, 4 }; 2 positions:



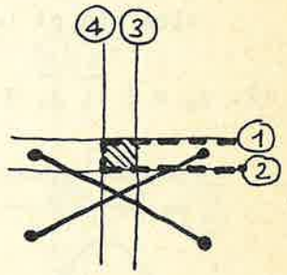

no subsidiary test: n(e,p) := 0;

14. $c_e$ = { 1, 3, 4 }; 2 positions:



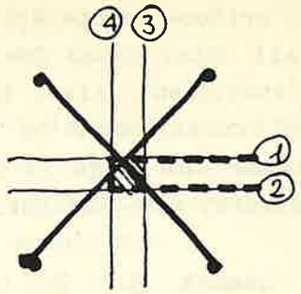

if ($c_1$={ 3 }) or ($c_2$={ 3 }) then n(e,p) := $\pm$0.5
else n(e,p) := 0;

15. $c_e$ = { 2, 3, 4 }; 2 positions:




if ($c_1$={ 3 }) or ($c_2$={ 3 }) then n(e,p) := $\pm$0.5
else n(e,p) := 0;

16. $c_e$ = { 1, 2, 3, 4 }; 2 positions:




```
if (x1 = abs(y1)) and (x2 = abs(y2))
     then Pclass(p,A) := .on.
else if (x1 > abs(y1)) and (x2 > abs(y2))
     then n(e,p) := 1
else if (x1 < abs(y1)) and (x2 < abs(y2))
     then n(e,p) := 0
else begin
     x := (yp - y2) (x1 - x2) +  x2
                    (y1 - y2)
     if xp = x  then Pclass(p, A) := .on.
     else if xp < x  then n(e,p) := 1
     else n(e,p) := 0;
     end;
```

**Remark 1.** The sum of positions grouped under every edge-code value is effectively 45; all possible relative positions of query point and polygon edges are consequently taken into account. This confirms the robusteness of the algorithm.

**Remark 2.** The proposed algorithm has to perform

some arithmetic operation only in the case $c_e = \{1,2,3,4\}$; in all other cases few logical and comparison operations are sufficient. Also, in the case $c_e = \{1,2,3,4\}$ it is not required any test on the nullity of the determinant, because the edge is not, in this case, parallel or coincident with the horizontal line for the test point.

**Remark 3.** The treatment of the case $c_e = \{1,2,3,4\}$ is consequent to the geometrical facts illustrated in figures 6a, 6b and 6c. The presence or the absence of an intersection point can be discovered with only comparison operations in subcases showed in figures 6a and 6b; at the contrary, one geometric intersection is needed for cases as in figure 6c.

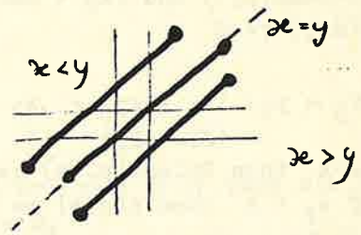



figure 6a

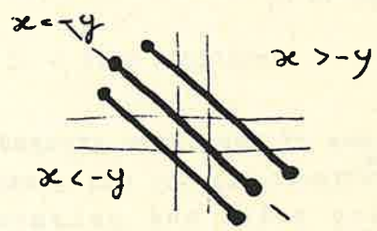



figure 6b

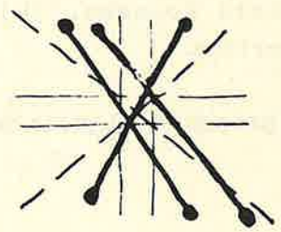


figure 6c

## 5. FORMALIZATION

In this section we give a formal description of the algorithm, making use of notations previously introduced. Before going to write formally the proposed algorithm we have to say something more about the assumption

$n(e,p) = \pm 0.5$.

The position is obvious when two consecutive edges $e_i = \langle u, v\rangle$ and $e_{i+1} = \langle v, w\rangle$ share their common vertex with the semi-infinite line for the test point; in this case we pose:

$n(e_i\ ,p) := 0.5$;
$n(e_{i+1},p) := 0.5$ if u and w don't lie inthe same horizontal subspace;
$-0.5$ otherwise.

The case of a sequence of edges lying on the semi-infinite line, and obviously preceeded and followed by two edges beeing in the above situation (see figure 5), is a little more complicated, but is resolvable in the same way, making use of a state variable and processing the edges in the circular order of the boundary.

The Polygon A is described as an ordered couple <P, >, being P the set of vertices and a permutation of P. We assume that contain only one cycle. The algorithm is immediatly extendable to the general case by extracting a cycle at a time from the given polygon.

```
Algorithm CLASSIFICATION   input:    p = [x   y]: point;
                                     A = <P,   >: polygon;
                           output:   v ∈ {.in., .on., .out.}

begin
    SET_TILE_BOUNDS;
    intersection_count := 0;
    crossing_status := ∅;
    pp ∈ P;  p1 := pp;  p2 := pp;
    c2 := TILECODE( p2 );
    if c2 = ∅ then v := .on.;
    while   ( p1 )) ≠ pp  and  v ≠ .on.
    do begin
        p1 := p2;   p2 :=   ( p1 );
        c1 := c2;   c2 := TILECODE( p2 );
        c_u := c1   c2;
        c_i := c1   c2;
        ce := c_u - c_i;
        TEST_EDGE_CODE( n(e,p), v );
        intersection_count := intersection_count + n(e,p);
    end;
  if v ≠ .on. then
    if intersection_count is odd then v := .in.
                                 else v := .out.
end.
```

The algorithm TEST_EDGE_CODE essentially consists of a Case structure testing the possible values of edge code $c_e$ by performing subsidiary tests specified in the previous section.

```
Algorithm TEST_EDGE_CODE
global variables:   ce,c_i,c_u,crossing_status: tile_code;
                    p,p1,p2: point;
          output:   n(e,p): integer;
                    v ∈ {.in., .on., .out.}
    begin
        n(e,p) := 0;
        if ce = ∅ then begin
            if c_u = ∅  then  Class(p, A) := .on. end
        else if ce = {1} then begin
            if c_i = ∅  then  Class(p, A) := .on.
            else if c_i = { 3 } then CROSSING_TEST end
        else if ce = {2} then begin
            if c_i = ∅  then  Class(p, A) := .on.
            else if c_i = { 3 } then CROSSING_TEST end
        else if ce = {3} then
        ....
        ....
        else if ce = {1,2,3,4} then
            if (x1 = abs(y1)) and (x2 = abs(y2))
                then Pclass(p,A) := .on.
            else if (x1 > abs(y1)) and (x2 > abs(y2))
                then n(e,p) := 1
            else if (x1 < abs(y1)) and (x2 < abs(y2))
                then n(e,p) := 0
            else begin
                x := (yp - y2) (x1 - x2) +  x2
                                (y1 - y2)
                if xp = x  then Pclass(p, A) := .on.
                else if xp < x  then n(e,p) := 1  end;
    end.
```

## 6. ANALYSIS OF THE ALGORITHM

The computational complexity of the presented algorithm is O(n) with respect to the number of logical operations, and O(q) with respect to the number of arithmetic operations, being n the number of polygon vertices and q the number of intersections between the horizontal semi-infinite line for the query point and the polygon boundary. The linearity in n is a theoretical bound for algorithms based on intersection counting or on angles summation; the efficiency improvement of the algorithm presented in this paper consists essentially in substituting arithmetic operations with comparison operations.

The efficiency of the procedure TEST_EDGE_CODE in the average case can be improved by using a sequence of "ELSE IF ($c_e$ = <value>)" tests linearly ordered in function of the decreasing expected frequence of edge-code values. In this way the most frequent cases are evaluated with the minimum number of confrontations.

In the following we assume that polygon vertices are uniformly distributed in the plane, i.e. vertices coordinates $x_i$ and $y_i$ are independent casual variables distributed with constant density. This assumption corresponds to perform a sort of worst case analysis, because, in the most of real cases, polygon edges have little dimensions with respect to the query area (the domain of the test point p) and the polygons are simple (edges intersect only at vertices). Under the above assumptions, a frequency analysis of edge-code values can be attempted as follows.

In a computer system the arithmetic of real numbers is approximate by a discrete arithmetic: let N be the cardinality of the finite set of "real" numbers that the computer system is able to represent. Imagine also that the query area coincides with the computer representation of the plane $\mathbf{R}_2$, having $N^2$ elements.

It is a very simple task to establish upper space bounds for the plane partition T(p) induced by any point p: diagonal tiles can have at most $N^2$ elements; cross tiles can have at most N; the central tile coincide with p. Formally we have:

$$
\begin{aligned}
|t_{11}| = |t_{13}| = |t_{31}| = |t_{33}| &= O(N^2);\\
|t_{12}| = |t_{21}| = |t_{23}| = |t_{32}| &= O(N);\\
|t_{22}| &= O(1).
\end{aligned}
$$

For a pictorial illustration of this property see figure 6.

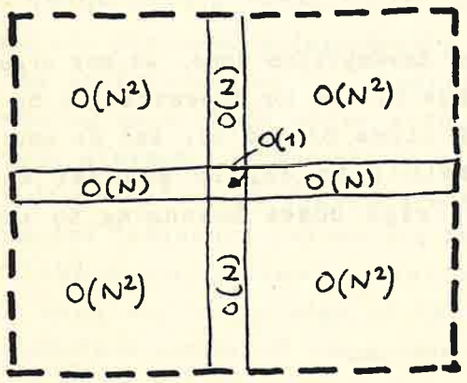



**Figure 6.** Space upper bounds for a tile partition of the plane.

We are now able to distinguish edge codes in function of the worst case cardinality of the tiles in wich their end-points fall. In other words, we set up a partial ordering of edge codes according to the rule:

$$c_e(\ e_i\ ) > c_e(\ e_j\ ) \quad \text{if} \quad C(v_1^i)C(v_2^i) > C(v_1^j)C(v_2^j),$$

being $C(v)$ equal to the exponent of the polynomial expression of the cardinality of the tile to which the vertex v belongs. With this rule three classes of edge codes with different expectation can be distinguished:

a) $C(v_1)C(v_2) = 4 \qquad [O(N^2),\ O(N^2)]$:
$c_e = \emptyset,\ \{1,2\},\ \{3,4\},\ \{1,2,3,4\}$;

b) $C(v_1)C(v_2) = 3 \qquad [O(N^2),\ O(N)]$:
$c_e = \{1\},\ \{2\},\ \{3\},\ \{4\},$
$\{1,2,3\},\ \{1,2,4\},\ \{1,3,4\},\ \{2,3,4\}$;

c) $C(v_1)C(v_2) = 2 \qquad [O(N^2),\ O(1)],[O(N),\ O(N)]$:
$c_e = \{1,3\},\ \{1,4\},\ \{2,3\},\ \{2,4\}$;

With the assumptions done, we may argument that edge codes of class a) can be expected to be more frequent than that of class b) and c), and so on. Nevertheless, none is possible to say in general about relative frequence of edge codes belonging to the same class; therefore, an optimal total ordering of tests cannot be defined, as largely application and data dependent.

For example, in the implementation of the algorithm presented in the Appendix 2, and oriented towards solid modeling applications, we make the further assumptions that edges are usually little with respect to the query area and that the query point is more frequently coincident with a polygon vertex than with an edge point.

## 7. CONCLUSION

In this paper we have presented an improved linear algorithm to compute point/polygon classification function. The algorithm is very robust, because it is based on the explicit consideration of any kind of possible input. In particular, polygon edges are partitioned in 16 disjoint classes, everyone caracterized by a different value of the "edge-code", and edges of different classes are processed independently.

Finally, the performance of the algorithm has been improved by coupling each input class with an expected value of frequency, evaluated in the hypothesis of uniform and independent distribution of polygon vertices in the plane. Such expected frequency values are used with the aim of establishing an optimal ordering for the computation; in such way every edge of the polygon is processed with a minimum number of comparisons.

It should be not too difficult to write a parallel version of the presented algorithm, working over the set

of the couples of adjacent edges and using a technique of composition of couples with a common horizontal edge.

## 8. REFERENCES


[1] Tilove, R.B., "Set Membership Classification: A Unified Approach to Geometric Intersection Problems", **IEEE Trans. on Computers, C-29**, 10, Oct. 1980, 874-883.

[2] Giloi, W.G., **Interactive Computer Graphics**, Prentice-Hall, Englewood Cliffs, N.J., 1978.

[3] Lee, D.T. and Preparata, F.P., "Computational Geometry - A Survey", **IEEE Trans. on Computers, C-33**, 12, Dec. 1984, 1072-1101.

[4] Shamos, M.I., "Geometric Complexity", in **Proc. 7th ACM Annu. Symp. Theory Comput.**, May 1975, 224-233.

[5] Chazelle, B.M. and Dobkin, D.P., "Detection is Easier than Computation", in **Proc. 12th ACM Annu. Symp. Theory Comput.**, May 1980, 146-153.

[6] Paoluzzi, A., "A New Algorithm for Polygon Clipping", **Report of "Dip. di Informatica e Sistemistica**, University of Rome "La Sapienza", Rep. 85-06, Jul. 1985.


## APPENDIX 1

In this appendix we present a Pascal implementation of the algorithm. User defined data type and procedures SET_TILE and TILECODE are omitted, as extremely simple. This implementation is presented in order to give a quantitative basis to the discussion of the algorithm performance in some real cases (see appendix 2).

```
procedure CROSSING_TEST( new,old: tile_code;
        var count: real; var status: tile_code);
  begin
    if status = [] then begin
      status := new;
      count := count + 0.5; end
    else begin
      if status = old then count := count + 0.5
                      else count := count - 0.5;
      status := []; end;
  end;


procedure CLASSIFICATION( p: point; pol: polygon;
                        var p_in, p_on, p_out: boolean );
  var c1,c2,c_un,c_int,c_edge,status: tile_code;
      b: box; i: integer; count: real;
      p1,p2,p_int: point;

begin
  p_in := false; p_on := false; p_out := false;
  with b do begin
    xmin := p.x; ymin := p.y;
    xmax := p.x; ymax := p.y; end;
  SET_TILE( b );
  count := 0; status := [];

  with pol do begin
    p2 := vert[n_vert]; i:= 0;
    TILECODE( p2, c2 );
    if c2 = [] then p_on := true;
    while (i < n_vert) and (not p_on) do begin
      i  := i + 1;
      c1 := c2;  p1 := p2;  p2 := vert[i];
      TILECODE( p2, c2 );
      if c2 = [] then p_on := true;
      c_un := c1 + c2;  c_int := c1 * c2;
      c_edge := c_un - c_int;

      if c_edge = [] then begin
        if c_un = [] then  p_on := true;  end
      else if c_edge = [3,4] then begin
        if c_un = [3,4] then p_on := true; end
```

```
else if c_edge = [1,2] then begin
  if c_int = [3] then count := count+1
  else if c_int = []  then p_on := true; end

else if c_edge = [1,2,3,4] then begin
  if (p1.x = abs(p1.y)) and (p2.x = abs(p2.y))
    then p_on := true
  else if (p1.x > abs(p1.y)) and (p2.x > abs(p2.y))
    then count := count + 1
  else if (p1.x < abs(p1.y)) and (p2.x < abs(p2.y))
    then count := count
  else begin
    p_int.x := ((p.y - p2.y)*(p1.x - p2.x)/
                (p1.y - p2.y)) + p2.x;
    if p_int.x > p.x  then count := count + 1
    else if p_int.x = p.x then p_on := true; end
  end
else if c_edge = [1,3,4] then begin
  if (c1 = [3]) or (c2 = [3]) then
    CROSSING_TEST( [1],[2], count,status ); end
else if c_edge = [2,3,4] then begin
  if (c1 = [3]) or (c2 = [3]) then
    CROSSING_TEST( [2],[1], count,status ); end
else if c_edge = [1,2,3] then count := count + 1
else if c_edge = [1] then begin
  if c_int = [] then p_on := true;
  if c_int = [3] then
    CROSSING_TEST( [1],[2], count,status ); end
else if c_edge = [2] then begin
  if c_int = [] then p_on := true;
  if c_int = [3] then
    CROSSING_TEST( [2],[1], count,status ); end
else if c_edge = [3] then begin
  if c_un = [3] then p_on := true; end
else if c_edge = [4] then begin
  if c_un = [4] then p_on := true; end
else if c_edge = [1,3] then begin
  if (c1=[]) or (c2=[]) then p_on := true
  else CROSSING_TEST( [1],[2], count,status ); end
else if c_edge = [2,3] then begin
  if (c1=[]) or (c2=[]) then p_on := true
  else CROSSING_TEST( [2],[1], count,status ); end
else if c_edge = [1,4] then begin
  if (c1=[]) or (c2=[]) then p_on := true end
else if c_edge = [2,4] then begin
  if (c1=[]) or (c2=[]) then p_on := true end;
  end; {case}
end; {with}
if not p_on then
  if odd( round( count )) then p_in  := true
                          else p_out := true;

end;
```

## APPENDIX 2

**Example 1.**

Consider the polygon with 21 vertices and the query point showed in figure 7; in this case the point/polygon classification function is determined by performing the following computational tasks:

a) evaluation of 21 tile codes, corresponding to nearly 63 confrontations among real numbers;

b) 21x3 = 63 boolean operations over 4-bit codes, to compute c_u, c_i, $c_e$;

c) 58 logical confrontations among 4-bit codes. As a matter of fact, 14 edges (bolded in the figure) are evaluated at the 1st confrontation, 4 edges at the 2nd, 4 edges at the 3rd, performing in this way 13(2) + 4(1 + 2) + 4(2 + 3) = 58 logical confrontations.

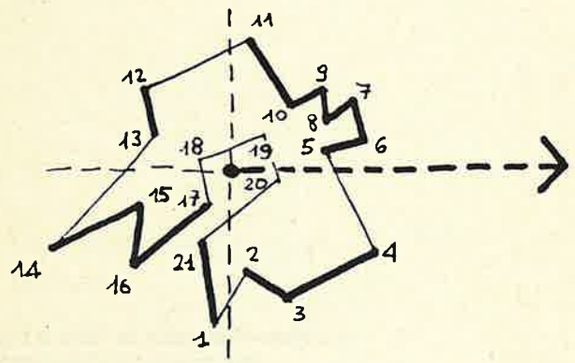



**Figure 7.** Example of query with a polygon of 21 vertices.

Finito di stampare nel mese di Luglio 1986
dall'Artigiana Multistampa Snc
Via Ruggero Bonghi, 36 - Roma

UNIVERSITÀ DEGLI STUDI DI ROMA
"LA SAPIENZA"

DIPARTIMENTO DI
INFORMATICA E SISTEMISTICA

A. PAOLUZZI

# A ROBUST, TILE-BASED ALGORITHM FOR POINT/POLYGON CLASSIFICATION

RAP. 03.86 GIUGNO 1986